COMP 4980 “Introduction to NLP”

Vector semantics

Stan Szpakowicz

Emeritus Professor

School of Electrical Engineering & Computer Science, University of Ottawa

TRU, Fall 2017

Dan Jurafsky’s slides (thanks!), adapted by Stan Szpakowicz

The plan

Back to counting

An old slide (Distribution, the fast way)

The much cited *distributional hypothesis*, "You shall know a word by the company it keeps!" (Firth 1957), underlies corpus-based *lexical semantics*,* usually known as *distributional semantics*.

We do not need to represent directly what a word means. It is enough to show this word in an interconnected network of words. Words which occur in a similar way in a (very) large corpus are quite likely to have similar meaning. And such a corpus better be searched really fast.

In essence, context restricts meaning. Consider these contexts:

- He drove his ☉ into a ditch.
- He poured a glass of ☉ into the bowl

*Lexical semantics is the study of what words mean and how they are related.

Why look for similar words? Question answering

Word similarity — or more specifically synonymy — has many uses in NLP.* Although there are advanced methods of computing such similarity, we will only discuss a few intuitions.

Question answering is an NLP task which benefits from similarity information.

For example, the words "tall" and "height" have similar distribution in any large corpus of English. That is to say, they must be similar in meaning.

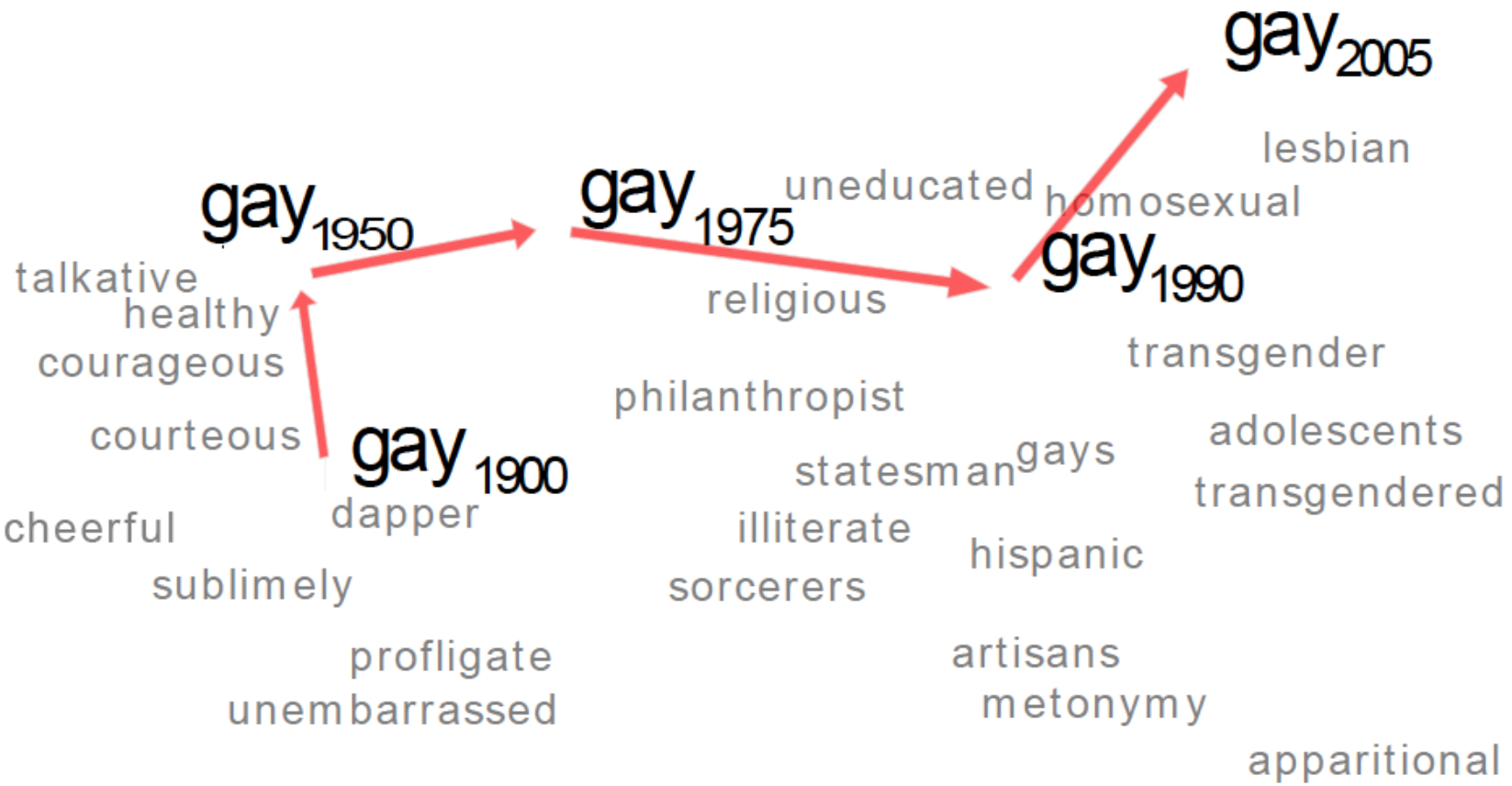
Question: "How tall is Mt. Everest?"

Candidate answer: "The official height of Mount Everest is 29029 feet."

*We will return to synonymy when we take another peek at WordNet.

Why look. . . ? Historical linguistics

Another NLP task based on the similarity of words and phrases is **historical linguistics**.*



*The picture comes from this paper. Note how words distributionally close to "gay" change.

Why look. . . ? Plagiarism detection

Yet another task: **plagiarism detection**.
The fragments in red are close enough to be interchangeable.
The fragments in black are essentially the same.

MAINFRAMES

Mainframes are primarily referred to large computers with rapid, advanced processing capabilities that can execute and perform tasks equivalent to many Personal Computers (PCs) machines networked together. It is characterized with high quantity Random Access Memory (RAM), very large secondary storage devices, and high-speed processors to cater for the needs of the computers under its service.

MAINFRAMES

Mainframes usually are referred those computers with fast, advanced processing capabilities that could perform by itself tasks that may require a lot of Personal Computers (PC) Machines. Usually mainframes would have lots of RAMs, very large secondary storage devices, and very fast processors to cater for the needs of those computers under its service.

Vectors?

A distributional model of meaning is usually represented by a vector-space model.* Thus: vector semantics.

> A bottle of tesgüino is on the table
> Everybody likes tesgüino
> Tesgüino makes you drunk
> We make tesgüino out of corn.

We also have these contexts:

> A bottle of beer is on the table
> Everybody likes beer
> Beer makes you drunk

So: tesgüino might be an alcoholic beverage.

*Things are represented by vectors. Two things are similar it their vectors are close by some metric.

Term-document matrix

A cell contains the count of term t in document d, a natural number. There are V terms and D documents.

A column represents a document: it is a vector in $\mathbb{N}^V$.

A row represents a word: it is a vector in $\mathbb{N}^D$. Each document is a context.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

So, for example, [1, 2, 37, 6] might represent "As You Like It". [37, 58, 1, 5] might represent "fool".*

*There would be many more documents and many (many) more terms.

Term-document matrix (2)

Two documents or two terms are similar if their vectors are reasonably close.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

So, *e.g.,* [8, 12, 1, 0] and [15, 36, 5, 0] are close. One popular similarity measure, on the scale 0 to 1, says 0.980.

That might be expected: "Julius Caesar" and "Henry V" are both Shakespeare's war-heavy history plays.

But it is 0.0477 for [8, 12, 1, 0] and [1, 2, 58, 117]. *Indeed: we have two plays in different Shakespearian genres.*

Term-document matrix (3)

Two documents or two terms are similar if their vectors are reasonably close.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

The vectors [37, 58, 1, 5] and [6, 117, 0, 0] are rather close. The same measure says 0.867.

Right: the words "fool" and "clown" appear in Shakespeare's comedies more often than in history plays.

Now take vectors [37, 58, 1, 5] and [2, 2, 12, 36]. The similarity is 0.146.

Correct: the words "fool" and "soldier" are not similar, and they tend to appear in different genres.

Word-word or word-context matrices

	aardvark	computer	data	pinch	result	sugar	. . .
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

. . .

We do not need complete documents for word similarity. Short contexts, maybe a paragraph or a window of several words to the left and to the right, should be quite enough.

A square matrix stores the counts of word-word co-occurrence.

A matrix can have a hefty 50,000 rows or something of this order, and a similar number of columns – - nothing like this 4×6 slice ☺.

Word-word or word-context matrices (2)

	aardvark	computer	data	pinch	result	sugar	. . .
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

. . .

Here is how contexts with a ± 7 window (ignoring punctuation) might look:

sugar , a sliced lemon, a tablespoonful of	apricot	preserve or jam , a pinch each of
their enjoyment . Cautiously she sampled her first	pineapple	and another fruit whose taste she likened
was well suited to programming on the	digital	computer . In finding the optimal R-stage policy
for the purpose of gathering data and	information	necessary for the study authorized in the

Most of the cells would be zeroes, but such *sparse* matrices are efficiently implemented.

More on word-word matrices

Shorter windows, $\pm$1-3 words, capture syntactic facts well. Windows of $\pm$4-10 words help make more semantic observations.

First-order co-occurrence (**syntagmatic association**): the two words are usually near each other.

wrote is a first-order associate of *book* or *poem*

Second-order co-occurrence (**paradigmatic association**): the two words have similar neighbours.

wrote is a second-order associate of *said* or *remarked*

The plan

How alike are they?

Cosine similarity

Cosine similarity is the cosine of the angle between the vectors:

$$cos\_sim(x, y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}}$$

For example:

$cos\_sim([8, 12, 1, 0], [15, 36, 5, 0]) \approx 0.979889587335047$.

This measure is sometimes presented as distance (that is to say, dissimilarity):

$$distance(x, y) = 1 - cos\_sim(x, y)$$

In the example, $\sim$0.020103 means very close vectors.

Pointwise mutual information

PMI between two words answers this question:

do words w_1 and w_2 co-occur more often than they would have if there were independent?

$$PMI(w_1, w_2) = log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

PMI ranges from -∞ to +∞. Negative values would suggest degrees of "unrelatedness". That is something we do not care for, nor do we understand it well enough.

We only consider half of PMI, known as positive PMI:

$$PPMI(w_1, w_2) = max(log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}, 0)$$

Computing PPMI

We have a matrix $\mathcal{F}$ with W rows (for words) and C columns (for contexts).*

Element $\mathcal{F}_{ij}$ counts occurrences of word w_i in context c_j.

Let us define a few sums first.

$$rowSum_i = \sum_{j=1}^{C} \mathcal{F}_{ij}$$

$$columnSum_j = \sum_{i=1}^{W} \mathcal{F}_{ij}$$

$$matrixSum = \sum_{i=1}^{W} \sum_{j=1}^{C} \mathcal{F}_{ij}$$

* W and C may be equal for all we care.

Computing PPMI (2)

Now, probabilities.

$$p_{ij} = \frac{\mathcal{F}_{ij}}{matrixSum}$$

$$p_{i*} = \frac{rowSsum_i}{matrixSum}$$

$$p_{*j} = \frac{columnSum_i}{matrixSum}$$

$$pmi_{ij} = log_2 \frac{p_{ij}}{p_{i*}\ p_{*j}}$$

$$ppmi_{ij} =$$

if $pmi_{ij} > 0$: pmi_{ij}

else: 0

Computing PPMI (3)

Raw frequency counts

	computer	data	pinch	result	sugar
apricot	0	0	1	0	1
pineapple	0	0	1	0	1
digital	2	1	0	1	0
information	1	6	0	4	0

$$rowSum = [2, 2, 4, 13]$$
$$columnSum = [3, 7, 2, 5, 2]$$
$$matrixSum = 19$$

$$p(w = information, c = data) = 6/19 \approx 0.316$$
$$p(w = information) = 11/19 \approx 0.579$$
$$p(c = data) = 7/19 \approx 0.368$$

Computing PPMI (4)

Probabilities (rounded!)

	computer	data	pinch	result	sugar	$p(word)$
apricot	0.000	0.000	0.053	0.000	0.053	0.105
pineapple	0.000	0.000	0.053	0.000	0.053	0.105
digital	0.105	0.053	0.000	0.053	0.000	0.211
information	0.053	0.316	0.000	0.211	0.000	0.579
$p(context)$	0.158	0.368	0.105	0.263	0.105	± 1.000

$$p(information, data) \approx log_2(0.316/(0.368 * 0.579)) \approx 0.569$$

$$p(digital, result) \approx log_2(0.053/(0.368 * 0.211)) \approx -0.551$$

Computing PPMI (5)

PPMI values (rounded!)

	computer	data	pinch	result	sugar
apricot	-	-	2.248	-	2.248
pineapple	-	-	2.248	-	2.248
digital	1.663	0.000	-	0.000	-
information	0.000	0.566	-	0.467	-

The matrix has *many* zeroes (no contexts for a given word). Even in this toy example half of the cells are zeroes.

PMI is biased toward infrequent events. Very rare words have very high PMI values. We could:

- give rare words slightly higher probabilities;
- apply Laplace smoothing (which has a similar effect).

Boost to rare context words

$$PMI_\alpha(w, c) = log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}$$

$$P_\alpha(c) = count(c)^\alpha \; / \; \sum_c count(c)^\alpha$$

If $\alpha < 1.0$ and $P(c)$ is small, then $P_\alpha(c) > P(c)$. $\alpha = 0.75$ has been found particularly effective.

Suppose that there are two events: $count(a) = 99$ and $count(b) = 1$. Obviously, $P(a) = 0.99$ and $P(b) = 0.01$.

$$P_{0.75}(a) = 99^{0.75} \; / \; (99^{0.75} + 1^{0.75}) \approx 0.969$$

$$P_{0.75}(a) = 1^{0.75} \; / \; (99^{0.75} + 1^{0.75}) \approx 0.031$$

Smooth the table

Add-2 Laplace smoothing

	computer	data	pinch	result	sugar
apricot	2	2	3	2	3
pineapple	2	2	3	2	3
digital	4	3	2	3	2
information	3	8	2	6	2

Probabilities (rounded!)

	computer	data	pinch	result	sugar	$p(word)$
apricot	0.034	0.034	0.051	0.034	0.051	0.203
pineapple	0.034	0.034	0.051	0.034	0.051	0.203
digital	0.068	0.051	0.034	0.051	0.034	0.237
information	0.051	0.136	0.034	0.102	0.034	0.356
$p(context)$	0.186	0.254	0.169	0.220	0.169	± 1.000

Smooth the table (2)

Smoothed PPMI values (rounded!)

	computer	data	pinch	result	sugar
apricot	0.000	0.000	0.561	0.000	0.561
pineapple	0.000	0.000	0.561	0.000	0.561
digital	0.616	0.000	0.000	0.000	0.000
information	0.000	0.583	0.000	0.375	0.000

Unsmoothed PPMI values (rounded!)

	computer	data	pinch	result	sugar
apricot	-	-	2.248	-	2.248
pineapple	-	-	2.248	-	2.248
digital	1.663	0.000	-	0.000	-
information	0.000	0.566	-	0.467	-

Document similarity

Another worthy NLP task:

calculate the similarity between documents.

The basic idea is uncomplicated.

1. Take a set of documents. Find terms: lemmatized content words are a typical set of terms.
2. Build a term-document matrix. Calculate frequencies.
3. Apply clever normalization.
4. Build a document-document matrix of cosine similarity values.

The measure often employed in this task is tf-idf —

term **f**requency modified by **i**nverse **d**ocument **f**requency.

tf-idf

Let t_i be a term, d_j a document. Let D be the document set.

Term frequency tf_{ij} can be simply the frequency f_{ij} of t_i in d_j. It can be more complicated, too.

- Occurrence: $tf_{ij} = 1$ if t_i is in d_j, 0 otherwise.
- Frequency adjusted for document length:

$$tf_{ij} = f_{ij} \,/\, (|d_j|).$$

- Logarithmically scaled frequency: $tf_{ij} = log(1 + f_{ij})$

The intuition behind inverse document frequency:

> a term which occurs in few documents distinguishes those documents well from the rest of the collection; a term which occurs frequently across the collection is unhelpful.

tf-idf (2)

Let $|D|$ be the total number of documents, and df_i the number of documents in which t_i occurs.

A popular definition of idf is

$$idf_i = log \frac{|D|}{df_i}.$$

The logarithm helps "squash" the typically high values in any realistic collection.

In the end, the weight of t_i in d_j is

$$w_{ij} = tf_{ij}\ idf_i$$

In effect, tf-idf prefers words frequent which are

frequent in d_j,

rare overall in the collection.

tf-idf (3)

The definition of idf may need an adjustment for toy problems, when the frequencies are low, often zeroes. Maybe like this:

$$idf_i = log \frac{|D| + 1}{df_i + 1} + 1.$$

This is how it is done in our practical example, adapted from Luling Huang's post (thanks).

The plan

That's it

On to a little more semantics

Bear with me. ☺